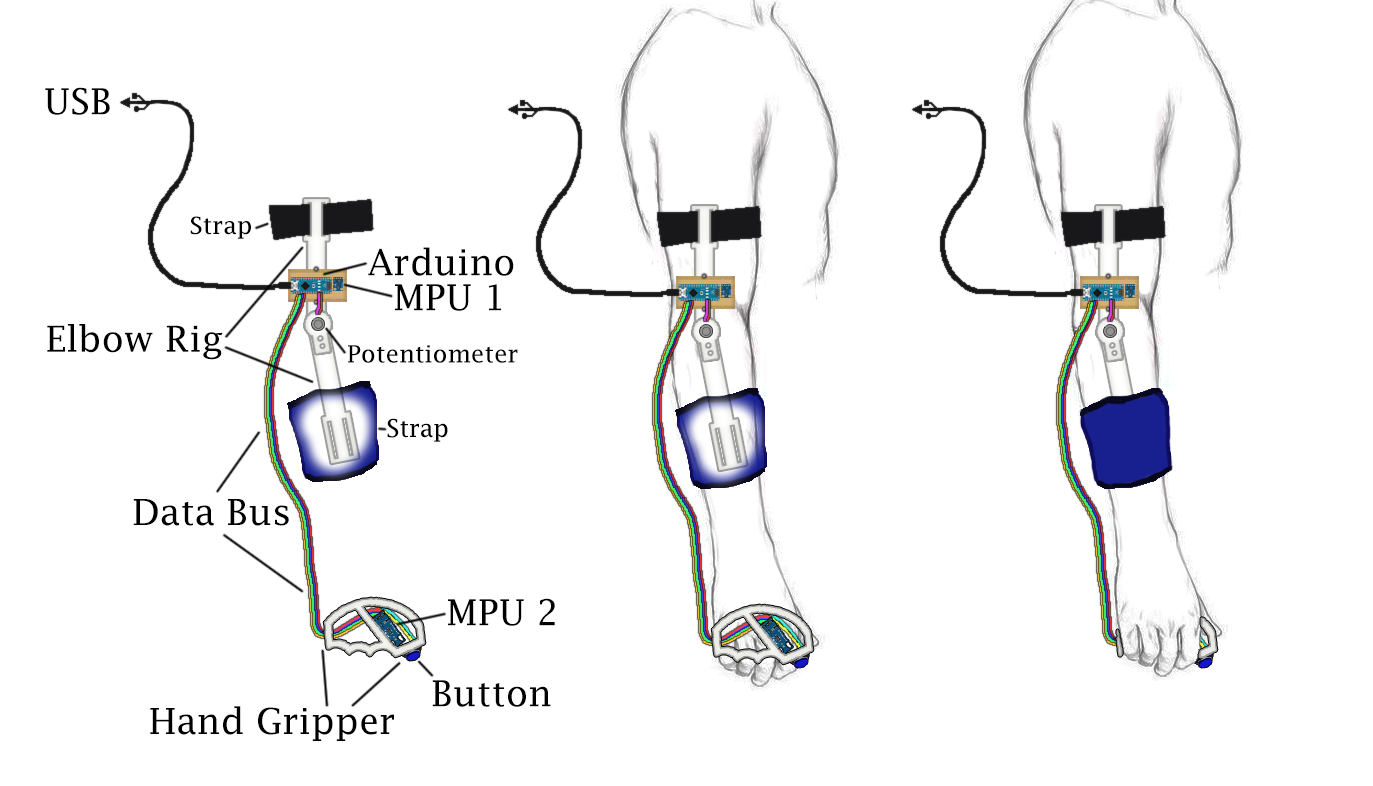
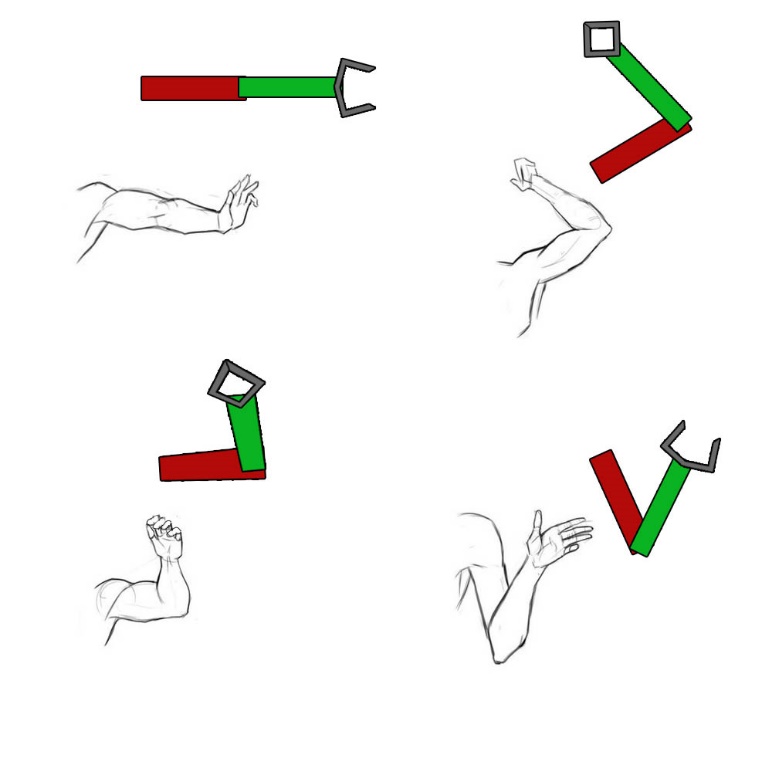
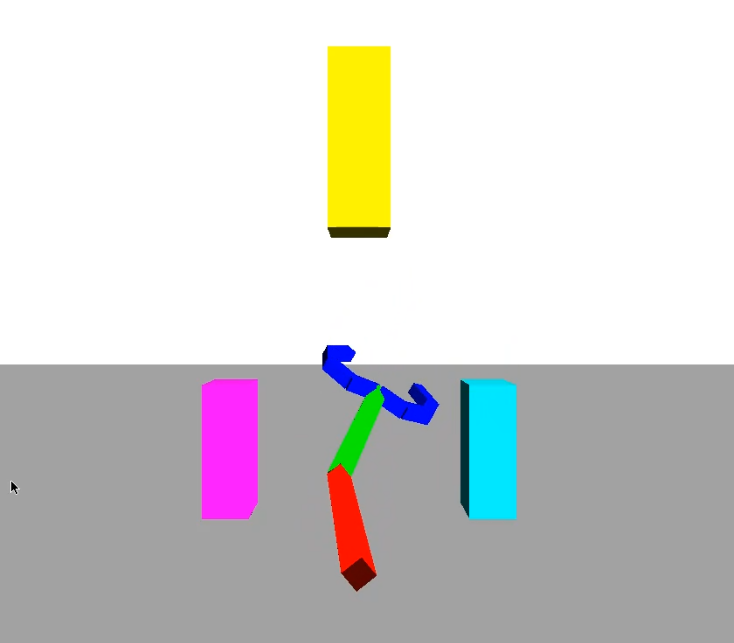
**Summary:**

The system I developed is a real time arm motion capture rig using custom Arduino hardware rendered using OpenGL in a python environment. The system comprises three distinct parts, the physical rig, its embedded software on the Arduino, and the rendering software running on the PC. The rig has two gyro/accelerometer sensors for the shoulder and hands, a potentiometer for the elbow joint, and a button at the end of the hand held part of the rig. The Arduino software interprets and packages the sensor data and passes it through a USB interface to the PC. The PC software renders the virtual arm and environment, as well as interpreting in real time the rig data being sent over the serial connection to update the 3D space.

This system is intended to solve problems of real time (or with slight modification, general purpose) motion capture. This problem is especially relevant now with the rise of VR popularity, and my gyro/accelerometer usage is very similar to how many motion tracking controllers work. I have been interested in Arduino as a hobby for years now, and integrating that with animation has lead me through interesting problems with the system, and has helped me understand the material of this course to a much greater degree.

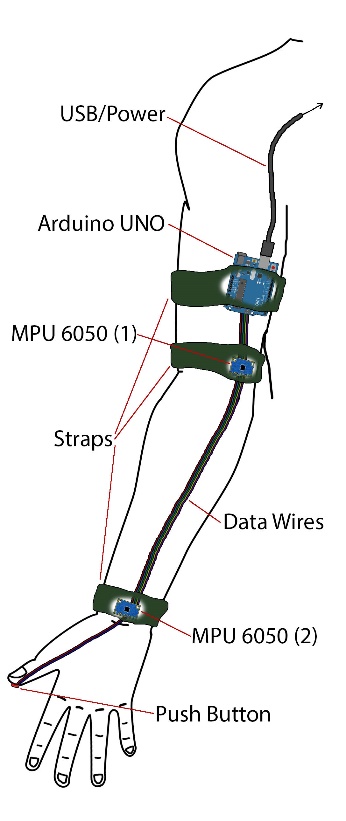






**Description of Work:**

The work was broken up into three parts, each with its own intrigue and challenges:

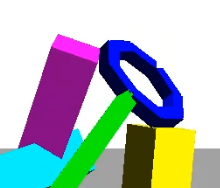
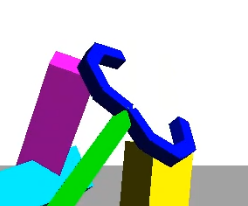
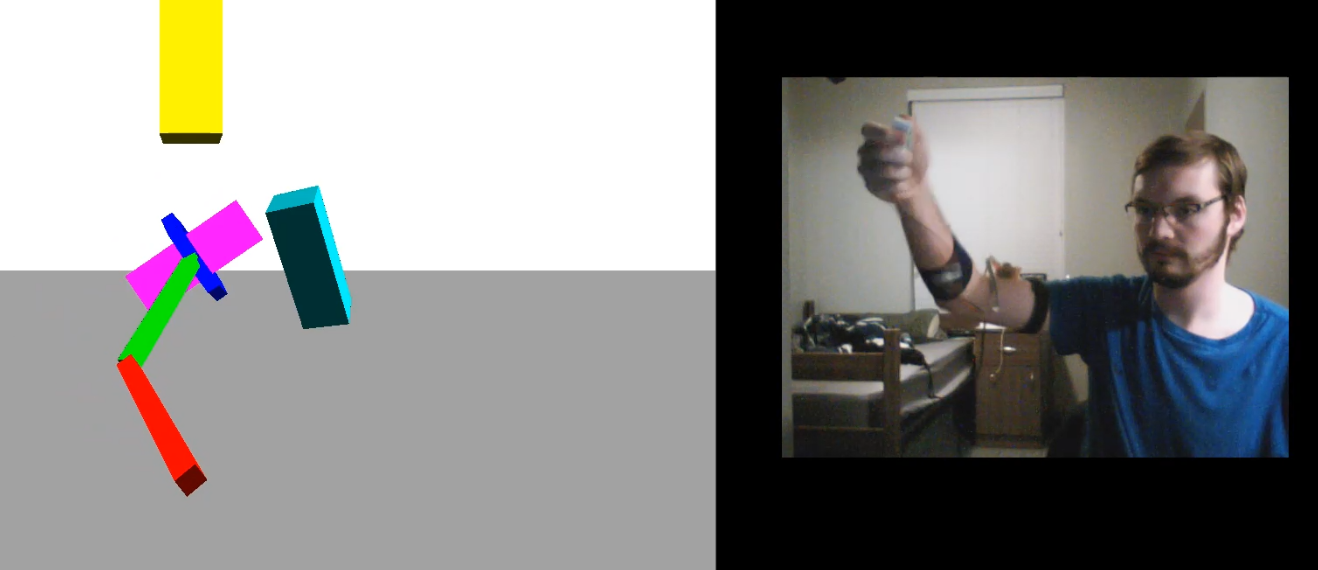
* Physical Rig

Original, obsolete, design

* + Initially the physical rig was going to be just two gyro/accelerometer sensors, but by one of my friends, John Bonnin, had access to a 3D printer and convinced me to go with a more complex arm rig.
  + Also originally the plan was to use a raspberry pi for the brain, or at least the rendering station. However it proved to be too susceptible to overheating as well as, ironically, not having enough power to run.
  + The new arm rig has three major parts:
    - The Arduino Board
      * The Arduino board contains the Arduino Nano and one of the MPU6050 Gyro/Accel sensors.
      * It is soldered together and screwed into the joint rig.
      * It contains all the routing wires that connect together all the circuitry
    - The Potentiometer Joint
      * This part was designed and printed by my friend using Google Sketchup and was printed on his Anet A8, and features a movable joint with a linear variable resistor to measure the rotation angle.
      * Because the elbow joint has only one degree of freedom, it can be sensed with a single potentiometer instead of using an entire new 3 DOF MPU6050
      * This part contains the second MPU6050 Gyro/Accel sensor as well as a push button on its end.
      * The button gives the user a grip input instead of just an orientation/position input that the MPU6050 gives.
  + Problems / Challenges
    - The Arduino Board
      * The Arduino and sensors were originally going to be just attached via bread board, but this was unreliable and so the rig was soldered down.
      * The tethering wire ribbon connecting the hand gripper to the Arduino board was too loosely connected, and had to be permanently soldered down.
    - The Potentiometer Joint
      * The joint can pop off its hinge if given a sufficient off axis torque. This can be avoided by being careful. A sturdier joint could not be printed to do various limitations.
      * The potentiometer can get rotated all the way to one of its maxes if not adjusted after the joint pops off.
    - The Hand Held Gripper
      * This piece actually has no issues that I know of, apart from the MPU6050 senor issues described in the Embedded Software section.
* Embedded Software
  + The software running on the Arduino is a heavily modified version of Jeff Rowberg’s i2cdevlib example for the MPU6050 - <https://github.com/jrowberg/i2cdevlib>
  + The Arduino reads quaternion data from both MPU6050s, the voltage across the joint rig’s linear variable resistor, and the state of the button to send along the serial connection.
  + Problems / Challenges
    - Using the generic serial connection some characters cannot sent through the USB without disrupting communication, so raw float data needed to be encoded such that it cannot generate illegal characters. This involved taking every byte of data to be sent and turning it into two bytes, each containing half the original byte. Padding the other half of the new bytes ensures the data sent are always readable legal characters. The receiving software simply does the opposite to decode the data.
    - The MPU6050 has a fault in its design that breaks I2C communication standards that causes an infinite loop beyond my control. What triggers the error is unknown, but the bug is documented and I was unware of this until late in development. So after about 10 seconds the Arduino will soft lock and stop sending serial data, freezing the PC program.
    - Due to how gyro/accelerometer sensors work there is always a slight rotation drift along the y-axis. This problem cannot be fixed within the scope of this project, it would require either a precise compass sensor (doesn’t exist) or an external sensing tower/cameras that can give relative yaw to the rig. This is how other motion capture rigs that use accelerometers work, see the Oculus, Vive, and others.
* OpenGL Python PC Software
  + The display is served using pygame - <https://github.com/pygame/>
  + The environment and rig are rendered using pyOpenGL - <https://github.com/mcfletch/pyopengl>
  + The OpenGL format and orientation parts are similar to what we’ve done for class assignments, but done in python instead of C++.
  + The software reads serial data from USB and decodes it into quaternion / sensor data, then converts those into rotation matrices to be used for OpenGL calculations.
  + Problems / Challenges
    - Quaternions are weird.
      * The MPU6050 sensors send four values (w, x, y, z) that should form the quaternion, and we learned how to turn quaternion data into rotation matrices in class.
      * However, no combination of the values sent from the sensors form correct orientations; at least one rotation axis are always either negated or orthogonal to what they should be.
      * To translate the sensor’s quaternion data to a correct virtual representation one has to, in this order:
        + interpret the incoming quaternion as (z, w, x, y)
        + form the rotation matrix as normal (henceforth referred to as R)
        + negate column 0 of R
        + negate column 2 of R
        + swap columns 0 and 2 of R
        + swap columns 1 and 2 of R
      * This bizarre formulation was found by pure trial an error on my part, and other similar projects that use the MPU6050 must do some equally byzantine math on the sensor data.
    - Due to how gyro/accelerometer sensors work the initial rotation about the y-axis is random; the sensor cannot know which way is forward to start with, only up and down due to acceleration from gravity. This means that the initial rotation needs to be zeroed by a constant offset yaw measure. This is also true for the hand MPU6050.
    - Because the hand’s MPU6050 rotation data is a world measurement, even though the hand is parented to the rest of the arm rig it cannot rotate with it. To achieve this I simply parented the hand as normal, then rotated it by the inverse of the current matrix to reset it to no rotation (but with correct translation), then apply the MPU6050 rotation for the local hand orientation.
    - The other objects in the space were probably the easiest to integrate, but despite my initial hopes I simply did not have time to add any physics or gamification to the final product.
    - Adding lighting was surprisingly one of the most challenging parts of this project. This isn’t because it’s hard or complex, but there isn’t a lot of clear documentation about what needs to be done to make simple lighting.
    - If I were to do this project over I would use Unity, or something other than OpenGL. It just doesn’t look good.

**Results:**

I made a motion capture rig that captures a person’s left upper/lower arm movement and orientation. This is represented in the demo program as a three bone aperture that can grab and move around three different boxes within reach. Video of the system working will be attached to this document for demonstration. The fatal flaws of the system are that it drifts in rotation about the y-axis over time, and will eventually freeze (approx. 10 seconds). See the Embedded Software problems and challenges section for details as to why.



**Analysis of Work:**

* New Results:
  + My results are not new, several implementations of arm motion capture have been done (most, better than mine sadly) by other amateurs:
    - <https://www.youtube.com/watch?v=9N3dccOZHAk>
    - <https://www.youtube.com/watch?v=oF5qok82e5I>
    - <https://www.youtube.com/watch?v=XF7klY2kiuQ>
  + My project however is the first that I have seen in the amateur space that uses a potentiometer for one of its axis. So while other rigs that have the same functionality as mine ultimately have redundant sensors (3 MPU’s compared to my 2).
  + If I could change my MPU6050 library (far too late for this project) to remove the freezing error, and figure out how other people in these examples seem to mitigate the y-axis drift so well, my rig could be fairly novel and useful.
* Meeting Goals:
  + Original Goals:
    - To have built a working piece of custom hardware that can be used for accurate motion capture.
    - To make the rig expandable so that more linkages can be added and more complex rigs can be made with little extra development.
    - ~~To have programed software that can be used to control and get data from such a rig for general purposes, i.e. to make a library for the hardware.~~
    - To make a demo that is portable and demonstrates the power of the rig.
  + I met most of the goals, and even the one I didn’t meet entirely is almost there. More importantly than those goals though I didn’t get to make the demo program nearly as interesting as I wanted.
  + Originally I had the idea of adding simple particle physics and just have the arm interact with spheres, but the problems and challenges stated in the above just made the complexity of the demo an afterthought.
  + Crippling problems like the drift and the freezing force out any notion of a game, and until I solve them, the rig cannot be used for much beyond demoing basic arm motion.